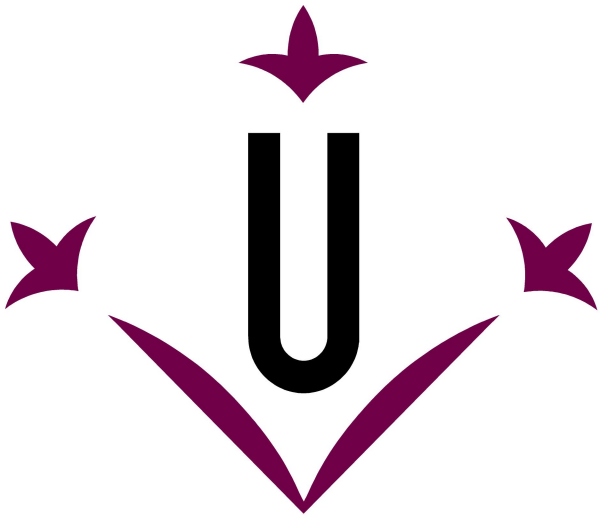

Universitat de Lleida

Desarrollo de aplicación web de retos para el tratamiento de pacientes con dolor crónico

Autor: Gerard Montilla Romia
Directora: Rosa Mª Gil Iranzo

Septiembre de 2017

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

En primer lugar, quisiera agradecer a mi familia todo el apoyo que me han brindado durante todos estos años. Sin su ayuda, nunca habría conseguido llegar a donde estoy ahora. Gran parte de mi éxito ha sido gracias a ellos.

En segundo lugar, a mi tutora, Rosa Mª Gil Iranzo, que a pesar de mis circunstancias personales, me ha brindado la oportunidad de llevar a cabo este proyecto y me ha aconsejado cuando lo he necesitado.

En tercer lugar, a todos y cada uno de los profesores de los que he disfrutado durante toda la carrera. Han sido capaces de despertar en mi un hambre de conocimiento del que creía que no disponía.

Por ultimo lugar, a mis compañeros de clase, con los que he pasado muy buenos momentos, aún estando a más de 1300 Km de distancia, como en este ultimo año.

Resumen

Esta aplicación pretende ser un complemento al Portal sobre Dolor Crónico en la que los usuarios puedan encontrar una serie de retos con los que divertirse a medida que interiorizan los contenidos ofrecidos por dicho Portal.

En este documento se detallará el desarrollo de una aplicación web para pacientes con dolor crónico. Se enumeraran y explicaran cada una de las fases en la construcción de esta aplicación empezando por el análisis, siguiendo por el diseño y terminando con el desarrollo.

Capítulo 1: Introducción

En este capítulo se detallarán los objetivos del proyecto, así como cual ha sido la causa que ha llevado a realizar este trabajo. Se detallará el tiempo que ha sido necesario emplear para llevarlo a cabo. Finalmente, se dará un presupuesto aproximado del desarrollo íntegro de la aplicación.

1.1. Objetivos

El principal objetivo del proyecto es diseñar y desarrollar una aplicación web que sirva para albergar diversas retos en forma de "minijuegosçuyo objetivo es reforzar los conocimientos ofrecidos en el Portal sobre Dolor Crónico . La aplicación web deberá:

- Permitir añadir nuevos retos a la plataforma.
- Permitir que los usuarios sean capaces de completar los retos.
- Permitir a los usuarios obtener recompensas al completar retos.
- Permitir a los usuarios comprobar sus logros en cada uno de los retos.
- Añadir una versión del juego del ahorcado[21] en forma de conjunto de retos para la plataforma.

Como objetivos secundarios y personales del desarrollo de la aplicación web se pueden enumerar los siguientes:

- Mejorar los conocimientos en desarrollo web.
- Trabajar con un *framework*[23] de desarrollo web moderno.

1.2. Motivación

En un principio se planteó la idea de crear una versión para dispositivos móviles del Portal sobre Dolor Crónico para la plataforma Android. Esa primera idea de proyecto venía dada del hecho de haber diseñado y desarrollado varias aplicaciones para dicha plataforma.

Más adelante, esa idea fue desechada puesto que se propuso crear un complemento para la web mencionada en el anterior párrafo. Dicha propuesta consistía en

crear una plataforma de juegos y retos a modo de actividad motivadora en la que se demuestre que el paciente esta integrando los principales mensajes de la actividad educativa. Esa idea provino del Vicedecano y Coordinador de Relaciones Internacionales de la Universitat de Lleida y parte del equipo del Portal sobre Dolor Crónico, el Dr. Fran Valenzuela Pascual.

Personalmente, el hecho de diseñar y desarrollar una aplicación que pueda servir para ayudar a otras personas es el factor más me motivaba en este proyecto.

1.3. Temporalidad

Para estimar el tiempo necesario para llevar a cabo el proyecto de forma satisfactoria se ha escogido dividir cada una de las fases de este en tareas. A cada una de las tareas se le asignara un cierto numero de días para llevarla a cabo. Esto permitirá dar un presupuesto mas o menos ajustado a la realidad.

Como se ha comentado en la sección 1.2, inicialmente, el objetivo del proyecto era el desarrollo de una aplicación para dispositivos Android. Las figuras 1.1 y 1.2 muestran el Diagrama de Gantt que se estableció antes del cambio de rumbo.

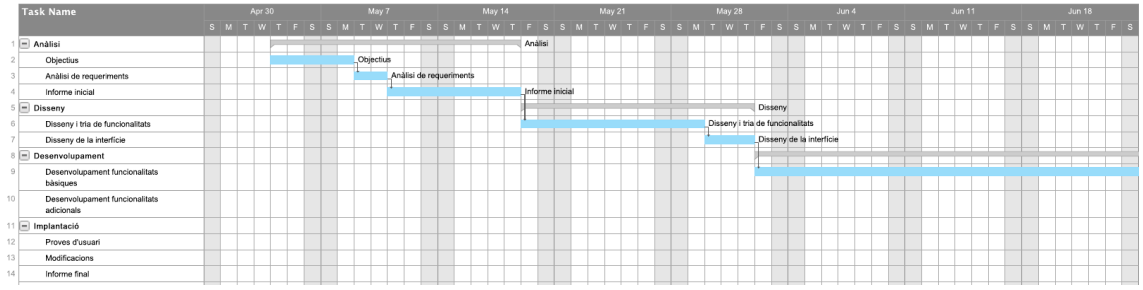

Figura 1.1: Diagrama de Gantt. Parte 1

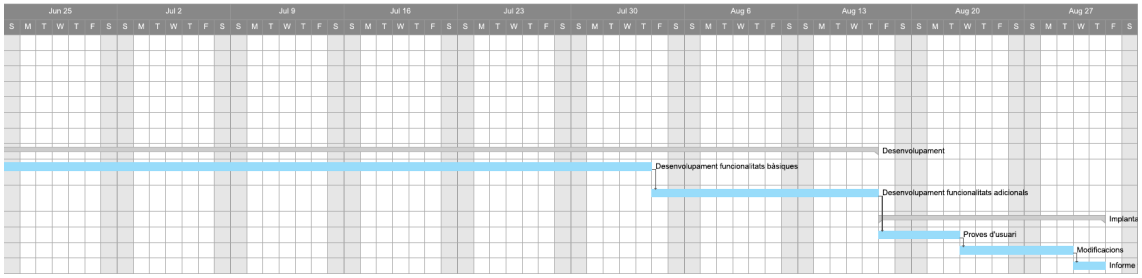

Figura 1.2: Diagrama de Gantt. Parte 2

En el cuadro 1.1 puede verse el Diagrama de Gantt de una forma mucho mas sencilla, con la fecha de inicio y de finalización de cada una de las tareas. Es necesario aclarar que esta planificación esta muy alejada del coste real en tiempo que ha sido necesario emplear, puesto que la idea que se propuso inicialmente y la que se materializó tras el cambio de rumbo eran muy diferentes.

Tarea -	**Fecha de inicio**	**Fecha de finalización**
Análisis	4/5/2017	18/5/2017
Objetivos	4/5/2017	8/5/2017
Análisis de requerimientos	9/5/2017	11/5/2017
Informe inicial	12/5/2017	18/5/20
Diseño	19/5/2017	1/6/2017
Diseño y elección de funcionalidades	19/5/2017	28/5/2017
Diseño de la interfaz	29/5/2017	1/6/2017
Desarrollo	2/6/2017	18/8/2017
Funcionalidades básicas	2/6/2017	4/8/2017
Funcionalidades adicionales	5/8/2017	18/8/2017
Implantación	19/8/2017	31/8/2017
Pruebas de usuario	19/8/2017	22/8/2017
Modificaciones	23/8/2017	29/8/2017
Informe final	30/8/2017	31/8/2017

Cuadro 1.1: Planificación de tareas inicial

En la tabla 1.2 se muestra el tiempo real que ha sido necesario emplear para llevar a cabo el proyecto. En la planificación inicial se disponía de un total de diecisiete semanas para la realización del proyecto. Tras el cambio de rumbo, se ha pasado a disponer de ocho semanas para el desarrollo del mismo.

Tarea	**Fecha de inicio**	**Fecha de finalización**
Análisis	25/7/2017	31/7/2017
Objetivos	25/7/2017	29/7/2017
Análisis de requerimientos	29/7/2017	31/7/2017
Diseño	31/7/2017	8/8/2017
Diseño y elección de funcionalidades	31/7/2017	3/8/2017
Diseño de la interfaz	3/8/2017	8/8/2017
Desarrollo	8/8/2017	12/9/2017
Funcionalidades básicas	8/8/2017	25/8/2017
Funcionalidades adicionales	25/8/2017	12/9/2017
Finalización	28/8/2017	8/9/2017
Informe	25/8/2017	8/9/2017

Cuadro 1.2: Planificación de tareas final

Debido a la falta de tiempo, se han eliminado varias de las tareas que se habían programado inicialmente. La fase de implantación, que englobaba las pruebas de

usuario no ha podido llevarse a cabo. Además, se ha decidido eliminar de las tareas de Análisis la parte de informe inicial por la misma razón.

Como se ha podido comprobar, cualquier imprevisto puede hacer que el proyecto no se lleve a cabo de la forma planificada, pudiendo incluso acabar en el fracaso de este.

1.4. Presupuesto

Para dar una estimación aproximada del coste del desarrollo, se han tenido en cuenta todos las herramientas empleadas y tiempo invertido en la aplicación. Se han buscado siempre herramientas de código abierto que permitiesen reducir la inversión necesaria. En el cuadro 1.3 se muestra el presupuesto desglosado.

Artículo	**Precio (€)**	**Cantidad**
Pencil	0	1
Atom	0	1
Inkscape	0	1
PC	1200	1
Desarrollador	12	250
Total	**4200**	

Cuadro 1.3: Presupuesto

En el cuadro 1.3 se ha obviado el precio del mantenimiento del software, puesto que únicamente se han tenido en cuenta las fases de diseño y desarrollo. Es un factor que habría que tener en cuenta cuando se despliegue la plataforma.

Capítulo 2: Gamificación

En este capítulo se explicará qué es la Gamificación, pues es la razón de existencia de esta aplicación.

2.1. Qué es la Gamificación

Tal como describe Virginia Gaitán[5], la Gamificación es una técnica de aprendizaje que traslada la mecánica de los juegos al ámbito profesional con el fin de conseguir mejores resultados. Dichas mejoras vienen dadas por el hecho de motivar a los participantes.

Antes que nada, es necesario buscar que es aquello que hace que el usuario mantenga la atención. Como bien explica Regina Nelson de la Universidad de Wisconsin-Platteville [14], si se consiguen identificar dichos elementos, pueden transferirse a otros entornos.

2.2. Elementos clave en los juegos

En el libro *"For the Win"*[20] de Kevin Werbach y Dan Hunter se describen los elementos clave de los juegos. Se organizan en una pirámide con tres categorías en los **Componentes** son la base, las **Mecánicas** son la parte central y las **Dinámicas** son la cúspide.

Comenzando por las Dinámicas, se describen cinco elementos que aunque no estén relacionados con el juego, deben ser considerados cuando se desarrolla un sistema de Gamificación:

- **Restricciones:** son las limitaciones que durante el proceso de diseño son acordadas.
- **Emociones:** las emociones son una parte inherente de los juegos. Es importante tener en cuenta que cada usuario es único.
- **Narrativa:** es la parte narrativa que dirige el juego. Puede captar el interés y generar vinculo entre usuario y juego.
- **Progresión:** muestra el avance del usuario en el juego a medida que el vinculo entre usuario y juego se hace mas fuerte.

- **Relaciones:** tiene en cuenta las interacciones sociales que ocurren cuando se participa en el juego o sistema, ya sean jugando con otros usuarios o solos.

El segundo grupo de elementos, conocido como Mecánicas, son los procesos que hacen posible que se establezca unión entre usuario y contenido. Se contemplan diez elementos:

- **Retos:** son las tareas que mueven al usuario a buscar una solución.
- **Oportunidad:** define el componente de aleatoriedad en el juego. Esta aleatoriedad puede despertar la curiosidad del usuario.
- **Competición:** es una mecánica muy común en juegos, en la que un jugador o grupo gana frente a otro que pierde.
- **Cooperación:** hace que los jugadores trabajen juntos con el fin de conseguir el objetivo común. Esta muy asociado con la competición.
- **Retroalimentación:** se refiere a la información que se le da al usuario sobre su progreso.
- **Adquisición de recursos:** a medida que el usuario progresa en el juego, obtiene recursos o coleccionables.
- **Recompensas:** son los beneficios que obtiene el usuario por completar alguna acción o reto.
- **Transacciones:** son los intercambios entre usuarios.
- **Turnos:** La participación en el juego de forma secuencial promueve el proceso de avanzar en el juego.
- **Estados de Victorias:** representa un elemento muy importante en los juegos que describe el objetivo que hace que un usuario sea el ganador.

En el fondo, cada Mecánica esta ligada con una o mas Dinámicas. Para aplicar correctamente las Dinámicas, es importante tener en cuenta que son únicamente en elemento mas de diseño en las herramientas de la Gamificación.

Por ultimo, los Componentes forman el grupo con más numero de elementos. Por lo general, estos elementos son mas concretos que los de los dos grupos superiores:

- **Hitos.**
- **Avatares.**
- **Insignias.**
- **Jefes finales.**
- **Colecciones.**
- **Combates.**

- **Contenido desbloqueable.**
- **Regalos.**
- **Tablas de clasificación.**
- **Niveles.**
- **Puntos.**
- **Búsquedas.**
- **Grafos sociales.**
- **Equipos.**
- **Bienes virtuales.**

Aunque no es una lista extensa, es una lista representativa de los diversos componentes del proceso de Gamificación.

2.3. Contextualización

Una vez enumeradas y explicadas las tres grandes categorías, se procederá a aplicarlas en el contexto de la aplicación. Para ello, se comenzará en el orden descrito en la sección 2.2.

2.3.1. Dinámicas

Los elementos que forman parte de las Dinámicas y que han tenido que tenerse en cuenta son los siguientes:

- **Restricciones:** puesto que el tiempo del que se dispuso era limitado, se acordó que únicamente se iba a crear un reto para la aplicación, la adaptación del juego del ahorcado.
- **Emociones:** las animaciones intentarán evocar emociones en los usuarios para avanzar en el conocimiento de su dolor.
- **Narrativa:** se mostrarán mensajes en cada uno de las actividades que ayudarán al usuario a centrar la atención en el reto.
- **Progresión:** se mostrará una barra de progreso en la que el usuario sabrá en todo momento cuantas actividades ha completado del reto en cuestión.
- **Relaciones:** una animación para cada actividad tratará de crear un vinculo entre usuario y juego.

2.3.2. Mecánicas

Puesto que este es un grupo de menor importancia que las Dinámicas, se han usado los siguientes elementos en el diseño y desarrollo:

- **Retos:** se crearán un conjunto de actividades asociadas a un reto.
- **Retroalimentación:** se mostrar una barra de progreso en los retos y en el perfil de cada usuario se mostrara las actividades completadas.
- **Recompensas:** se otorgará un diploma cuando se complete el reto del ahorcado.
- **Estados de Victorias:** cuando un usuario complete todos las actividades del reto del ahorcado, se le considerará ganador de ese reto.

2.3.3. Componentes

Por ultimo, del conjunto de elementos mas concreto de las tres categorías se han escogido los siguientes que se aplicarán en el juego:

- **Hitos:** conseguir terminar todas las actividades del reto del ahorcado.
- **Insignias:** diploma al completar el reto.

Capítulo 3: Estado del arte

En este capítulo se describirán las tecnologías relacionadas con la aplicación y los estudios que ha sido necesario realizar para su desarrollo.

3.1. Tecnologías relacionadas

En este apartado se describirán los componentes mas importantes de la aplicación, así como el software que se ha usado para el diseño y el desarrollo. En cada uno de los componentes y software se detallará el porqué de la elección, explicando los motivos que se tuvieron en cuenta para tomar cada una de las decisiones.

3.1.1. Back-end

En este apartado se enumerarán y explicarán los componentes mas importantes que forman parte del *back-end*.[1]

3.1.1.1. Node.js

Figura 3.1: Node.js

Node.js[15] es un entorno de ejecución multiplataforma de código abierto para la capa de servidor basado en el lenguaje de programación Javascript. Es asíncrono y usa un modelo de operaciones E/S sin bloqueo y orientado a eventos. Esta diseñado para construir aplicaciones en red escalables

Históricamente, JavaScript se ha usado principalmente para correr *scripts*[33] del lado del cliente, en los cuales dichos *scripts* están dentro del código HTML de la pagina web esperando a ser ejecutado por el motor de JavaScript del navegador del cliente. Node.js se ha convertido en uno de los defensores principales del paradigma *"JavaScript everywhere"*[2], permitiendo unificar el desarrollo web en un solo lenguaje de programación, en lugar de recalar en varios.

[1] Parte de servidor o capa de acceso a datos.
[2] JavaScript para todo.

3.1.1.1.1 Elección

El principal factor que ha hecho decantarse por Node.js para construir el *back-end* es el hecho de estar familiarizado con el entorno.

Como ventajas que nos aporta Node.js frente a otros es que un entorno muy minimalista y rápido. Su funcionalidad puede extenderse gracias a la gran cantidad de módulos que existen. Se trata de software libre distribuido bajo la licencia MIT[27]. Es muy eficiente, ya que permite que una maquina relativamente poco potente gestione una gran cantidad de conexiones. Al trabajar con JavaScript, homogeneiza el desarrollo *Full Stack*[3] bajo un único lenguaje de programación.

3.1.1.2. Express

Figura 3.2: Espress

Express[4] es un *framework* de desarrollo de aplicaciones web Node.js mínimo y flexible que proporciona un conjunto solido de características para las aplicaciones web y móviles. Es software libre que se distribuye bajo la licencia MIT. Es el *framework* estándar para servidor de Node.js. Ofrece los métodos necesarios para manejar peticiones HTTP[32] y un sistema simple de enrutamiento.

3.1.1.2.1 Elección

Se ha escogido Espress como *framework* porque se había trabajado con el anteriormente. Además, es el *framework* de desarrollo web de Node.js más utilizado, lo que implica mejor documentación.

3.1.1.3. Passport

Figura 3.3: Passport

Passport[17] es un *middleware*[28] para Node.js cuyo propósito es autenticar peticiones a través de un conjunto de *plugins*[31] conocidos con el nombre de estrategias. Dichas estrategias incluyen el uso de proveedores *OAuth*[16] tales como Facebook, Twitter o Google, aunque también provee de estrategias para el autenticado local a través de un nombre de usuario y una contraseña.

[3] *Back-end* y *Front-end.*

Cada una de estas estrategias están empaquetadas en módulos individuales disponibles en el gestor de paquetes de Node.js, lo que hace muy sencillo dar soporte a cada método diferente de autenticación.

3.1.1.3.1 Elección

Se ha escogido Passport porque es el *middleware* de autenticación con mejor documentación que se ha encontrado en el gestor de paquetes de Node.js. Es muy fácil de usar y muy modular.

3.1.2. Base de datos

En este apartado se enumeraran y explicaran los componentes mas importantes que forman parte de la base de datos.

3.1.2.1. MongoDB

Figura 3.4: MongoDB

MongoDB[11] es un sistema de base de datos NoSQL[30] orientado a documentos. Se distribuye como software libre bajo la licencia GNU AGPLv3[24]. En lugar de guardar datos en tablas como las bases de datos relacionales, MongoDB guarda estructuras de datos similares a JSON.

MongoDB esta diseñado para ser escalable, con un gran rendimiento y con alta disponibilidad de datos. Además, permite ejecutar código JavaScript del lado del servidor.

3.1.2.1.1 Elección

Se ha escogido MongoDB como base de datos porque hace al no tener esquema rígido, permite cambiar los datos si se requiere sin mayores complicaciones. Esta es la razón principal y es causada por la falta de tiempo. Únicamente se han contemplado funcionalidades básicas que quedan patentes en los modelos empleados para guardar datos en la base de datos.

3.1.2.2. Mongoose

Figura 3.5: Mongoose

Mongoose[12] es una herramienta de modelado de objetos para MongoDB diseñado para trabajar de manera asíncrona.

La ventaja de usar este tipo de herramienta es que añade una capa de abstracción. Es decir, se puede trabajar con la base de datos

en términos de JavaScript, en lugar lugar de trabajar en términos de la semántica de la base de datos.

3.1.2.2.1 Elección

Se ha elegido trabajar con Mongoose por el hecho de añadir una capa de abstracción con la base de datos. El hecho de trabajar con una herramienta de este tipo resulta en menor coste de desarrollo y mantenimiento de la aplicación.

3.1.3. Front-end

En este apartado se enumeraran y explicaran los componentes mas importantes que forman parte del *front-end*.[4]

3.1.3.1. JavaScript

Figura 3.6: JavaScript

JavaScript[10] es un lenguaje de programación interpretado de alto nivel, dinámico, débilmente tipado orientado a objetos y multiparadigma. Junto a HTML[26] Y CSS, es una de las tres tecnologías centrales de la producción de contenido en linea.

Se utiliza mayoritariamente en en el lado del cliente, permitiendo modificar el comportamiento de la pagina web. El código JavaScript es soportado por todos los navegadores, que se encargan de interpretarlo.

Además, junto a tecnologías como AJAX, permite enviar y recibir información del servidor sin necesidad de recargar la pagina.

3.1.3.1.1 Elección

Se ha escogido JavaScript porque sin ninguna duda es el lenguaje mas utilizado para el desarrollo web en la parte de presentación.

[4]Parte de cliente o capa de presentación.

3.1.3.2. Handlebars

Figura 3.7: Handlebars

Handlebars[8] es un sistema de plantillas semánticas web. Es una extensión de otro sistema de plantillas, Mustache[13], aunque añade algunos características.

Su principal objetivo es desacoplar la parte de presentación(HTML) de la parte de datos(JavaScript).

3.1.3.2.1 Elección

Se ha escogido Handlebars como sistema de plantillas porque resulta muy fácil generar paginas web dinámicas. Además, su sintaxis es muy parecida a HTML.

3.1.3.3. CSS

Figura 3.8: CSS3

CSS[19] es un lenguaje de diseño gráfico que sirve para dar estilo a la presentación de un documento estructurado escrito en lenguaje de marcado. Es muy usado para estableces el diseño de las paginas web e interfaces de usuario escritas en HTML.

Esta diseñado para marcar la separación del contenido del documento y la forma de presentación de este.

La especificación CSS es mantenida por el *World Wide Web Consortium (W3C)*, que es el organismo encargado de generar recomendaciones y estándares que aseguran el crecimiento de la web a largo plazo.

3.1.3.3.1 Elección

Se ha escogido CSS por que es un estándar de código abierto usado globalmente en la creación de contenido web.

3.1.3.4. Bootstrap

Figura 3.9: Bootstrap

Bootstrap[2] es un conjunto de herramientas de software libre para el desarrollo con HTML, CSS y JavaScript. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y una gran cantidad de elementos de diseño.

Si principal objetivo es adaptar la interfaz al tamaño del dispositivo en el que se visualiza el contenido web.

Como curiosidad, es el segundo repositorio con mas estrellas de GitHub.

3.1.3.4.1 Elección

Se ha escogido Bootstrap por la cantidad de elementos de diseño que ofrece, ayudando a crear interfaces sencillas, limpias, intuitivas y adaptadas a todo tipo de pantallas.

3.1.3.5. jQuery

Figura 3.10: Bootstrap

jQuery[18] es una librería JavaScript rápida, pequeña y con multitud de características. Permite el recorrido y la manipulación, manejo de eventos, animaciones y AJAX de una forma mucho mas simple.

Es software libre que se distribuye bajo las licencias MIT y GNU/GPLv2[25]. Ofrece una serie de funcionalidades que permiten ahorrar muchas lineas de código, con lo que se consiguen los mismo resultados con menos tiempo y menos código.

Algunos componentes de Bootstrap requieren del uso de la librería jQuery, por lo que ambos agentes tienen una estrecha relación.

3.1.3.5.1 Elección

Se ha escogido jQuery porque facilita la programación de la parte del cliente en gran medida, además, se han usado componentes de Bootstrap que requerían de este complemento.

3.1.3.6. AJAX

Figura 3.11: AJAX

AJAX[22] es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, mientras se mantiene la comunicación asíncrona con el servidor. De esta forma, es posible realizar cambios en la pagina sin necesidad de recargarla.

Es una tecnología asíncrona en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir en la pagina web ni en su comportamiento.

3.1.3.6.1 Elección

Se ha escogido AJAX porque era necesario enviar información al servidor una vez el usuario había finalizado correctamente el reto. Por tanto, AJAX encaja perfectamente en este aspecto.

3.1.4. Software

En este apartado se enumeraran los dos tipos de software que se han utilizado para el diseño y el desarrollo del proyecto.

3.1.4.1. Atom

Figura 3.12: Atom Editor

Atom[1] es un editor de texto de código abierto que se distribuye bajo la licencia MIT. Es personalízale hasta el mas mínimo detalle y tiene una interfaz moderna.

Tiene un gestor de paquetes integrado con el que es posible añadir infinidad de funcionalidades y cambiar la apariencia. Cuenta con soporte de control de versiones Git[6]. Esta desarrollado por GitHub[7].

3.1.4.1.1 Elección

Se ha escogido Atom por ser un editor de textos de código abierto. Es altamente personalizable y con la instalación de varios paquetes agiliza muchísimo el desarrollo, especialmente en la parte de front-end.

3.1.4.2. Inkscape

Figura 3.13: Inkscape

Inkscape[9] es un editor de gráficos vectoriales para la edición y creación de diagramas, gráficos, logotipos e ilustraciones complejas. Trabaja con archivos de tipo vectores de gráficos escalables o SVG. Se distribuye bajo la licencia GPLv2, por lo que se trata de software libre.

3.1.4.2.1 Elección

Se ha escogido Inkscape para la creación de las *sprite sheets*[34] de las animaciones por ser software libre y gratuito.

3.1.4.3. Pencil

Figura 3.14: Pencil

Pencil[3] es un software diseñado para el prototipado de interfaces gráficas. Es software libre distribuido bajo la licencia GPLv2.

3.1.4.3.1 Elección

Se ha escogido Pencil por ser software libre y gratuito, puesto que las demás alternativas que se encontraron eran todas de pago.

3.2. Estudios realizados

Para poder llevar a cabo este proyecto de manera satisfactoria, ha sido necesario la comprensión de la técnica de la Gamificación, explicada anteriormente en el capitulo 2.

Dicha técnica propone una serie de procesos para motivar al usuario con el fin de lograr un propósito.

El Portal sobre Dolor Crónico pretende conseguir que el usuario comprenda un poco más su dolor y no se sienta excluido del sistema sanitario. Para ello, es necesario motivarlo para que siga aprendiendo sobre las causas que le originan el dolor.

Aquí es dónde entra en juego la aplicación, cuyo fin es aplicar una serie de procesos de la Gamificación para crear un juego y conseguir mantener al usuario vinculado con la plataforma .

Capítulo 4: Desarrollo

En este capítulo se detallarán las diferentes fases por las que se ha pasado para la creación de la app web. Se empezará con el análisis, nombrando los requerimientos del proyecto. Se continuará con el diseño de las funcionalidades y de la interfaz y se terminará con el desarrollo de la funcionalidades.

4.1. Análisis

En este apartado se detallaran las requerimientos que se establecieron en la primera fase de desarrollo de la aplicación.

4.1.1. Análisis de requerimientos

4.1.1.1. Requerimientos funcionales

En el cuadro 4.1 se puede ver la todos los requerimientos funcionales que se establecieron junto con la descripción de cada uno de ellos.

Requerimiento	Descripción del requerimiento
RF1	La aplicación debe permitir el registro de nuevos usuarios y el acceso a los usuarios registrados.
RF2	La aplicación debe permitir añadir nuevos retos.
RF3	La aplicación debe permitir visualizar los retos.
RF4	La aplicación debe permitir al usuario completar los retos.
RF5	La aplicación debe almacenar el progreso de cada usuario en los retos.
RF6	La aplicación deber permitir al usuario visualizar los retos completados.
RF7	La aplicación debe permitir al usuario visualizar los diplomas obtenidos.
RF8	La aplicación debe permitir añadir nuevos diplomas.
RF9	La aplicación debe permitir visualizar los diplomas.
RF10	La aplicación debe permitir que el usuario obtenga diplomas cuando complete los retos.

Cuadro 4.1: Requerimientos funcionales

4.1.1.2. Requerimientos no funcionales

En el cuadro 4.2 que se encuentra mas abajo se puede ver la lista de requerimientos no funcionales junto con la descripción de cada uno.

Requerimiento	Descripción del requerimiento
RNF1	El ordenador que aloje la aplicación debe disponer de conexión a Internet.
RNF2	El ordenador que aloje la aplicación debe tener instalado Node y MongoDB.
RNF3	El ordenador que acceda a la aplicación debe disponer de conexión a Internet.
RNF4	La aplicación debe proporcionar tiempos de respuesta rápidos.
RNF5	La aplicación debe mantener los datos personales de los usuarios seguros.

Cuadro 4.2: Requerimientos no funcionales

4.2. Diseño

En este apartado se describirán las decisiones de diseño que se tomaron para construir la aplicación web.

4.2.1. Diseño y elección de funcionalidades

4.2.1.1. Base de datos

En la primera etapa de diseño se decidió empezar por el diseño de la base de datos. De esta forma, se establecen las entidades que servirán para modelar cada uno de los agentes que jugaran un papel activo en el software. Así pues tenemos los siguientes modelos en la aplicación.

4.2.1.1.1 User

Esta entidad se encarga de modelar a los usuarios que se registran y acceden a la aplicación. Tiene los siguientes campos:

- `id`: es un identificador único que representara cada instancia de `User` en la base de datos.
- `username`: es el nombre que el usuario escogerá cuando se registre y servirá para acceder mas adelante a la aplicación.
- `password`: es la contraseña asociada al nombre de usuario.

4.2.1.1.2 Activity

Este modelo representa cada una de las actividades que el usuario tendrá que superar y que estarán asociadas con un reto. Sus atributos son los siguientes:

- `id`: es un identificador único que representara cada instancia de `Activity` en la base de datos.
- `name`: es el nombre asociado a la actividad.
- `description`: es la descripción de la actividad.

4.2.1.1.3 Challenge

La entidad `challenge` modela un reto dentro de la aplicación, como se especificara mas adelante, este modelo tiene asociados actividades y diplomas. Estos son sus atributos:

- `id`: es un identificador único que representara cada instancia de `Challenge` en la base de datos.
- `name`: es el nombre asociado al reto.
- `diploma`: es el diploma que esta relacionado con el reto, de forma que cuando se completan todas las actividades del reto, se consigue el diploma
- `activities`: es el conjunto de actividades necesarias pera completar el reto.
- `description`: es la descripción de la actividad.

4.2.1.1.4 Diploma

Esta entidad representa el diploma que se obtiene al completar un reto. Sus atributos son los siguientes:

- `id`: es un identificador único que representara cada instancia de `Diploma` en la base de datos.
- `name`: es el nombre asociado al diploma.
- `description`: es la descripción del diploma.

4.2.1.1.5 UserActivity

Este modelo representa las actividades que el usuario ha completado, que son las que nos ayudaran a decidir si el usuario ha conseguido superar el reto. Sus atributos son los siguientes:

- `id`: es un identificador único que representara cada instancia de `UserDiploma` en la base de datos.
- `user`: almacena el `id` asociado al usuario.
- `activities`: guarda los `id` de cada reto que el usuario ha completado satisfactoriamente.

4.2.1.1.6 UserChallenge

Representa los retos que el usuario ha completado. Se usara para mostrar los logros de cada. Sus atributos son los siguientes:

- `id`: es un identificador único que representara cada instancia de `UserDiploma` en la base de datos.
- `user`: almacena el `id` asociado al usuario.
- `activities`: guarda los `id` de cada actividad que el usuario ha completado con éxito.

4.2.1.1.7 UserDiploma

Es una representación de los diplomas obtenidos por el usuario. Permitirá mostrar los diplomas obtenidos en el perfil de usuario. Tiene los siguientes atributos:

- `id`: es un identificador único que representara cada instancia de `UserChallenge` en la base de datos.
- `user`: almacena el `id` asociado al usuario.
- `activities`: guarda los `id` de cada diploma obtenido por el usuario tras completar un reto.

4.2.1.2. Back-end

Con las decisiones tomadas en el apartado 4.2.1.1 se procedió al diseño de la parte de servidor o *back-end*. Dicha parte es la encargada de hacer de intermediaria entre el cliente y los datos.

Así pues, se decidió que la parte de servidor debía permitir acceder y crear nuevas instancias de los objetos almacenados en la base de datos, dando lugar a los siguientes recursos.

4.2.1.2.1 Users

Este recurso deberá permitir lo siguiente:

- Registrar nuevos usuarios.
- Permitir el acceso de usuarios registrados.
- Obtener las diplomas, actividades y retos completados de cada usuario registrado.
- Denegar el acceso a cualquier usuario no registrado.

4.2.1.1.2.2 Activities

El siguiente recurso deberá:

- Registrar nuevas actividades.
- Obtener las actividades registradas en la base de datos.
- Denegar el acceso a cualquier usuario no registrado.

4.2.1.1.2.3 Challenges

El recurso `challenges` deberá:

- Permitir registrar nuevos retos.
- Obtener la lista de retos registrados en la base de datos.
- Denegar el acceso a cualquier usuario no registrado.

4.2.1.1.2.4 Diplomas

El recurso siguiente deberá:

- Registrar nuevos diplomas.
- Obtener todos los diplomas registrados en la base de datos.
- Denegar el acceso a cualquier usuario no registrado.

4.2.2. Diseño de la interfaz

Una vez establecidos los criterios en el diseño de la base de datos y el *back-end*, se pasó a diseñar la interfaz de usuario. Es necesario mencionar que para el diseño de la interfaz se tuvo en cuenta el diseño actual del Portal sobre Dolor Crónico.

En el diseño de la interfaz no se tuvieron en cuenta aspectos que fue necesario implementar durante el desarrollo, como todas las pantallas de registro de entidades y la pantalla inicial. Así pues, únicamente estarán descritos y explicados ciertos componentes que debido a la falta de tiempo se consideraron mas importantes.

4.2.2.1. Pagina inicial del reto

Es la pagina de inicio de un reto, puesto que uno de los objetivos del proyecto era la creación de una serie de actividades en las que el juego a implementar era el ahorcado.

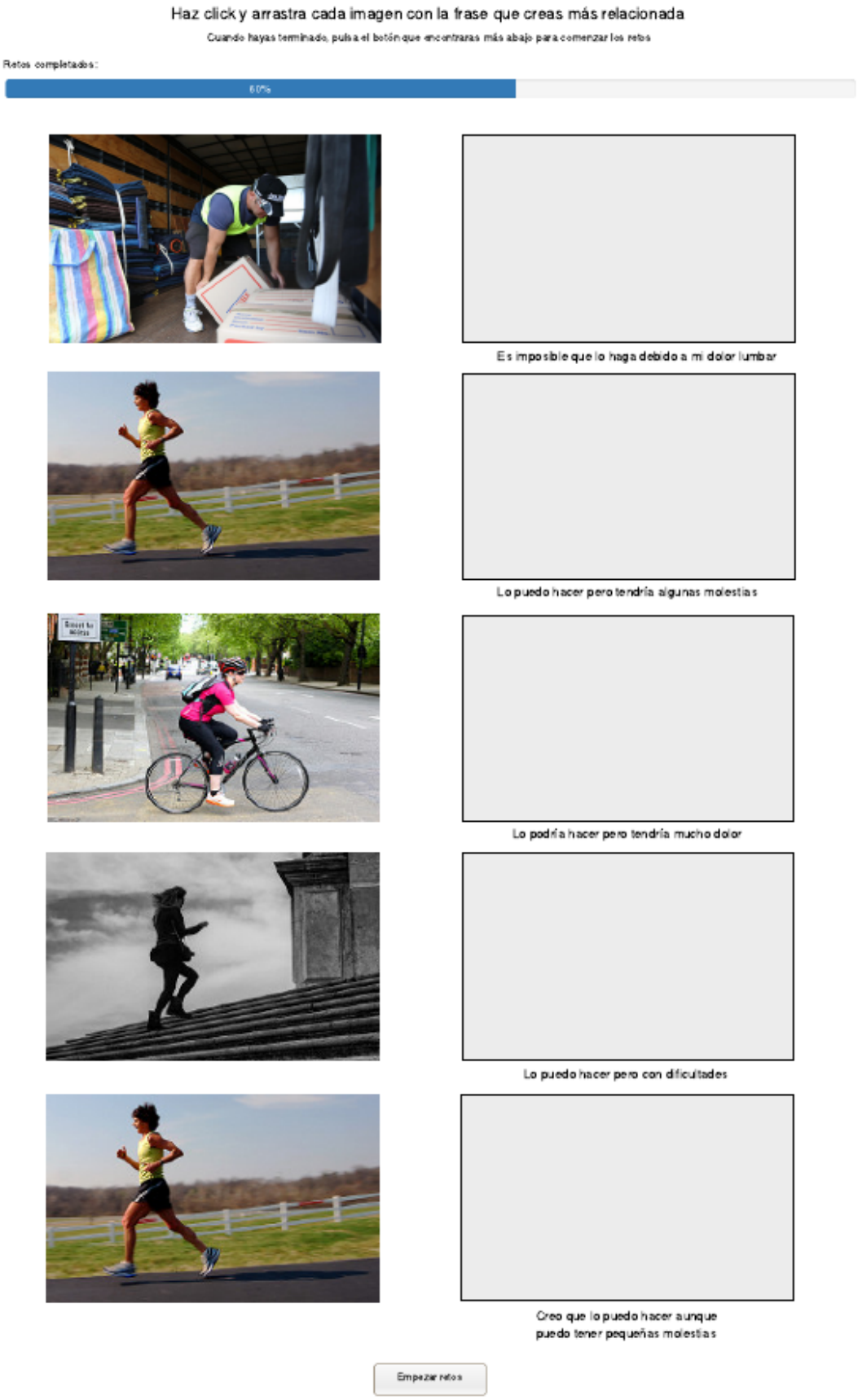

Figura 4.1: Pagina inicial reto del ahorcado

La figura 4.1 muestra el diseño de la pagina inicial del reto del Ahorcado. Esta formada por un conjunto de imágenes en movimiento que el usuario debe arrastrar hacia la frase con la que crea que haya mas relación. Tiene las siguientes partes:

- **Cabecera:** Muestra las instrucciones que el usuario debe seguir para empezar a participar en el juego. También muestra una barra de progreso en el que el usuario puede comprobar el porcentaje de actividades del reto completadas.
- **Cuerpo:** Como se ha mencionado anteriormente, contiene un total de cinco imágenes en movimiento que representaran cinco actividades. Dichas actividades deberán ser asociadas con las frases que el usuario considere más relacionadas.
- **Pie de pagina:** Muestra un botón que permitirá navegar hacia la primera actividad del reto.

4.2.2.2. Pagina del ahorcado

Es la página donde se podrá jugar al ahorcado. En la figura 4.2 se puede apreciar el diseño de la interfaz.

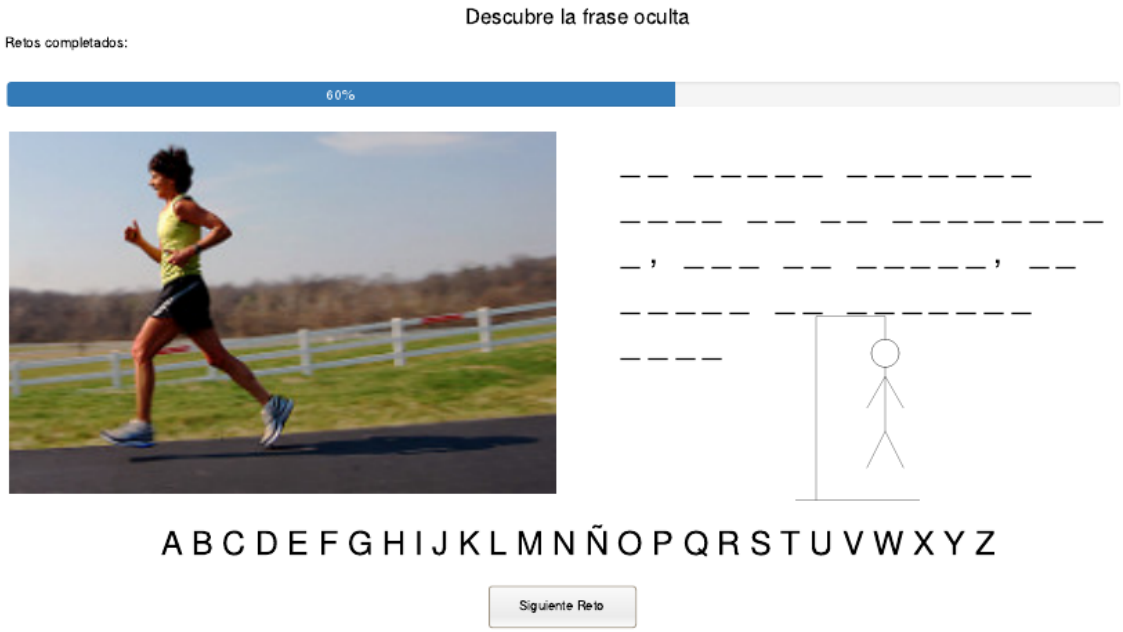

Figura 4.2: Pagina del ahorcado

Dicha página esta formada por varias secciones que a continuación van a ser enumeradas y explicadas:

- **Cabecera:** Muestra una pequeña pequeña descripción del objetivo de la actividad. Como en el caso del apartado anterior, muestra una barra con el progreso del usuario en el reto.

- **Cuerpo:** De izquierda a derecha, muestra una imagen en movimiento de una persona realizando una actividad física. A su derecha se presenta la frase oculta que el usuario debe descubrir, junto con el dibujo del ahorcado necesario para saber los intentos restantes. Por ultimo, debajo del todo muestra el abecedario en el que sera posible pulsar letras para descubrir la frase oculta.
- **Pie de pagina:** Muestra un botón que permitirá navegar hacia la siguiente actividad del reto.

4.2.2.3. Página de finalización de reto

La figura 4.3 muestra la pantalla que se mostrara cuando se hayan completado todas las actividades de un reto, en este caso, el ahorcado.

Figura 4.3: Página de finalización de reto

La pagina cuenta con la siguiente estructura:

- **Cabecera:** Muestra una breve descripción informando al usuario que ha completado todas las actividades. Como en las cabeceras de las anteriores paginas, también se mostrara una barra de progreso.
- **Cuerpo:** Contiene una imagen del diploma asociado al reto, contextualizado en el reto del ahorcado.
- **Pie de pagina:** Muestra un botón que permitirá navegar hacia todos los diplomas obtenidos.

4.2.2.4. Página de diplomas obtenidos

Por ultimo, en la figura 4.4 se muestra los diplomas que ha obtenido el usuario al completar los retos.

Figura 4.4: Página de diplomas obtenidos

Se pueden desglosar los siguientes elementos de su estructura:

- **Cabecera:** Muestra un texto informando de que se visualizan los diplomas obtenidos.
- **Cuerpo:** Contiene todos y cada uno de los diplomas que el usuario ha obtenido.

4.2.2.5. Diagrama

Una vez todos las paginas han sido descritas, se mostrara en la figura 4.5 el diagrama de funcionamiento de la aplicación.

Figura 4.5: Diagrama de funcionamiento

4.3. Desarrollo

En este apartado se reseñarán todas las decisiones, características o aspectos clave mas importantes en la ultima fase de desarrollo de la aplicación.

4.3.1. Funcionalidades básicas

4.3.1.1. Estructura

El primer paso para llevar a cabo el desarrollo del software era crear la estructura básica del proyecto. Para ello, se uso de la herramienta para crear proyectos que proporciona el paquete `express-generator` en el entorno Node.js. El comando que se uso para generar el proyecto es el siguiente:

- `express --hbs challenge`

Este comando genera todos las archivos y carpetas necesarios para tener una aplicación básica Node.js con el framework Express. Los argumentos del comando están explicados a continuación:

- `express:` es la herramienta de linea de comando para crear la estructura básica de una aplicación Express.
- `--hbs:` es el motor de plantillas que se usara para las vistas, en este caso Handlebars.
- `challenge:` es el nombre de la aplicación web que se creará.

Se decidió usar el patrón de arquitectura del software Modelo-Vista-Controlador (MVC)[29] ya que permite una separación clara entre la parte de representación y manejo de datos y la parte de interacción del usuario.

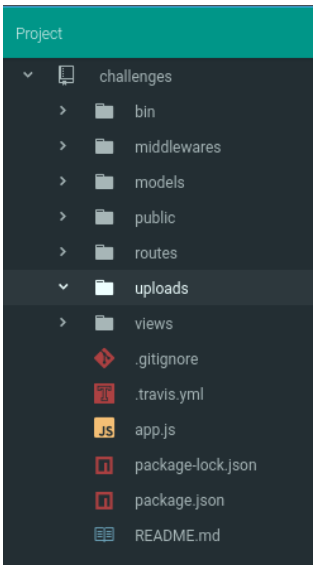

Figura 4.6: Estructura del proyecto

En la figura 4.6 puede observarse la estructura final del proyecto. Anteriormente, se ha mencionada que se iba a aplicar el patrón MVC. Hay que hacer remarcar que en la estructura mostrada en tal figura, el componente `routes` es el equivalente al componente controlador en el patrón Modelo-Vista-Controlador. Así pues, la función de cada uno de los componentes es la siguiente:

- `models`: contiene las representaciones de los modelos definidos en el apartado 4.2.1.1 y que van a permitir representar y almacenar la información en la base de datos.
- `views`: contiene todas las plantillas que serán renderizadas por la aplicación por medio del controlador, en este proyecto concreto, los archivos en el fichero `routes`.
- `routes`: almacena los ficheros que se encargar de procesar las peticiones web y ofrecer respuestas a estas. Contiene todas las rutas que la aplicación proporcionara y que serán accesibles.

4.3.1.2. Autenticación y registro de usuarios

La creación de un sistema de inicio de sesión y de registro de usuarios fue la primera parte que se desarrollo. Para poder almacenar información sobre el estado de las actividades, retos y diplomas completados de cada usuario era necesario crear un sistema de autenticación. Para este menester, se hizo uso del *middleware* Passport.

```
passport.use(new LocalStrategy (
  function(username, password, done) {
    User.getUserByUsername(username, function(err, user) {
      if (err) throw err;
      if (!user){
        return done(null, false, {message: 'Usuario desconocido'});
      }

      User.comparePassword(password, user.password, function(err, isMatch) {
        if (err) throw err;
        if (isMatch) {
          return done(null, user);
        } else {
          return done(null, false, {message: 'Contraseña incorrecta'});
        }
      });
    });
  }
));

passport.serializeUser(function(user, done) {
  done(null, user.id);
});

passport.deserializeUser(function(id, done) {
  User.getUserById(id, function(err, user) {
    done(err, user);
  });
});

router.post('/login', passport.authenticate('local',{
  successRedirect:'/', failureRedirect:'/users/login', failureFlash: true}), function(req, res) {
  res.redirect('/');
});
```

Figura 4.7: Acceso de usuarios registrados

La figura 4.7 muestra los métodos involucrados en el proceso de autenticación de usuarios registrados. el procedimiento de autenticación de usuarios es el siguiente:

1. Cuando se recibe un `POST` a `/users/login`, el método `passport.authenticate` es ejecutado con la estrategia escogida, en este caso `local`.
2. Passport toma los argumentos `req.body.username` y `req.body.password` y los pasa a la función de verificación de la estrategia.
3. Se busca el usuario en la base de datos y se comprueba si la contraseña introducido es la correcta.
4. Si el usuario es autenticado, se llama a la función `passport.serializeUser`, que se encarga de decir que información del objeto`user` es almacenado en la sesión, en este caso el identificador único de usuario.
5. Se redirecciona al usuario a la pagina raíz de la aplicación con el resultado de la operación anterior ajuntada en la petición como `req.user`.

En la figura 4.8 puede verse la pantalla de inicio de sesión. Dicha pantalla no se contemplo en el etapa de diseño, pero fue necesario crearla.

Figura 4.8: Pantalla de inicio de sesión

Para llevar a cabo el proceso de alta de nuevos usuarios en el sistema, el procedimiento es el siguiente:

1. Se recibe una petición de `POST` a `/users/register` que contiene el usuario y la contraseña, en este caso, la contraseña esta duplicada. Dichos argumentos pueden encontrarse en `req.body`.
2. Se comprueba que no existe un usuario en la base de datos del sistema con el mismo nombre y se comprueba que las dos contraseñas coincidan.
3. Se crea un nuevo objeto `User` con los atributos `usename` y `password` presentes en el cuerpo de la petición y se llama al método `User.createUser` que se encarga de escribir el usuario en la base de datos.

4. Se redirecciona al usuario a la ruta `/users/login` si todo ha salido correctamente, en caso contrario, se vuelve a mostrar la ruta `/users/register` con los errores encontrados.

En la figura 4.9 se muestra la parte mas importante de la función de registro de nuevos usuarios.

```
req.getValidationResult().then(function(result) {
  if (result.isEmpty()) {
    User.getUserByUsername(username, function(err, user) {
      if (err) throw err;
      else if (user) {
        var errors = [{param: 'username', msg: 'El usuario ya existe', value: username}];
        res.render('users/register', {
          errors: errors
        });
      } else {
        var new_user = new User({
          username: username,
          password: password
        });

        User.createUser(new_user, function(err, user) {
          if (err) return err;
          else {
            var new_userActivities = new UserActivities({
              user: user._id
            });
            var new_userDiplomas = new UserDiplomas({
              user: user._id
            });
            var new_userChallenges = new UserChallenges({
              user: user._id
            });
```

Figura 4.9: Creación de nuevo usuario

Por motivos de de seguridad obvios, todas las contraseñas de los usuarios son guardadas en la base de datos tras haber sido aplicadas un función de *hash* sobre ellas. En la figura 4.10 puede apreciarse el proceso de mapeado de la contraseña.

```
module.exports.createUser = function(newUser, callback) {
  bcrypt.genSalt(10, function(err, salt) {
    bcrypt.hash(newUser.password, salt, function(err, hash) {
      newUser.password = hash;
      newUser.save(callback);
    });
  });
};
```

Figura 4.10: Función de creación de nuevo usuario

El modelo que representa al usuario se encuentra dentro de la carpeta `/models/user.js`. Se puede ver su estructura en la figura 4.11.

```
// User Schema
var UserSchema = mongoose.Schema({
  username: {
    type: String,
    index:true
  },
  password: {
    type: String
  }
});

var User = module.exports = mongoose.model('User', UserSchema);
```

Figura 4.11: Modelo de `User`

El controlador que se encarga de gestionar cada uno de estos sucesos se encuentra en la ruta `/routes/users.js` del proyecto.

4.3.1.3. Registro y visualización de actividades

Tras tener un método que garantizaba la autenticación de usuarios funcional, el paso siguiente fue empezar a desarrollar el funcionamiento básico de la aplicación.

El primer paso en esa dirección fue generar la clase que modela una actividad, de esta forma, seria posible crearlas y visualizarlas .Se puede apreciar en la figura 4.12 el modelo `Activity`.

```
// Activity Schema
var ActivitySchema = mongoose.Schema({
  activityName: {
    type: String,
    index:true
  },
  description: {
    type: String
  }
});

var Activity = module.exports = mongoose.model('Activity', ActivitySchema);
```

Figura 4.12: Modelo de `Activity`

Para la creación de una nueva actividad es necesario estar autenticado. El procedimiento de creación de una actividad consta de los siguientes pasos:

1. Se recibe una petición de `POST` a `/activities/register` que contiene el nombre de la actividad y la descripción como parámetros en el cuerpo de la petición.

2. Se comprueba que no existe ninguna actividad con el mismo nombre que la presente en `req.body.activityName`. Si no es así, se vuelve a generar la vista mostrando como error que ya existe una actividad con ese nombre.

3. Se crea un objeto `activity` con los parámetros de `req.body` y se llama al método `Activity.createActivity`, que guarda en la base de datos la nueva actividad.

4. Se redirige al usuario hacia `/activities`, donde aparecerá la nueva actividad creada.

El controlador que se encargar de ofrecer estas funcionalidades se encuentra en la carpeta `/routes/activities.js`. La figura 4.13 muestra una parte de ester archivo.

```
// GET home page.
router.get('/', auth.ensureAthentication, function(req, res) {
  Activity.getActivities(function(err, activities) {
    if (err) throw err;
    else {
      res.render('activities/activities', {
        activities: activities});
    }
  });
});

// GET register page.
router.get('/register', auth.ensureAthentication, function(req, res) {
  res.render('activities/register');
});
```

Figura 4.13: Controlador de actividades

4.3.1.4. Registro y visualización de diplomas

Tras tener desarrollar la parte de actividades, el siguiente recurso que se creo fue el de diplomas. El recurso diplomas representa el premio asociado a la superación de un reto.
Para crear un nuevo diploma, los pasos son muy parecidos a los de la creación de una actividad. Es necesario estar autenticado para poder añadir o visualizar un diploma. El procedimiento esta a continuación enumerado.

1. Se recibe una petición de `POST` a `/diplomas/register` que contiene el nombre del diploma y la descripción como parámetros en el cuerpo de la petición.

2. Se comprueba que no existe ningún diploma con el mismo nombre que la presente en `req.body.diplomaName`. Si no es así, se vuelve a generar la vista mostrando como error que ya existe un diploma con el mismo nombre.

3. Se crea un objeto `diploma` con los parámetros de `req.body` y se llama al método `Diploma.createDiploma`, que guarda en la base de datos el nuevo diploma.

4. Se redirige al usuario hacia `/diplomas`, donde aparecerá el nuevo diploma creado.

En la figura 4.14 se muestra el esquema del modelo `Diploma` empleado en el recurso `/diplomas`.

```
// Diploma Schema
var DiplomaSchema = mongoose.Schema({
  diplomaName: {
    type: String,
    index:true
  },
  description: {
    type: String
  }
});

var Diploma = module.exports = mongoose.model('Diploma', DiplomaSchema);
```

Figura 4.14: Modelo de `Diploma`

4.3.1.5. Registro y visualización de retos

Tras tener construir un sistema capaz de registrar y autenticar usuarios y de crear y visualizar retos y diplomas, el ultimo paso para cubrir las funcionalidades básicas era la creación y visualización de retos. Puesto que en el diseño se consideró que un reto tiene asociado actividades y un diploma, el paso lógico era crear estas dos funcionalidades primero.

```
// Challenge Schema
var ChallengeSchema = mongoose.Schema({
  challengeName: {
    type: String,
    index:true
  },
  diploma: {
    type: Schema.ObjectId,
    ref: 'Diploma',
    required: true
  },
  activities: [{
    type: Schema.ObjectId,
    ref: 'Activity',
    required: true
  }],
  description: {
    type: String
  }
});

var Challenge = module.exports = mongoose.model('Challenge', ChallengeSchema);
```

Figura 4.15: Modelo de `Challenge`

En la figura 4.15 se muestra el esquema del modelo de la entidad reto, con su nombre en ingles, `Callenge`.

Para poder registrar un nuevo reto, hay que seguir un procedimiento muy parecido al registro de nuevos diplomas o actividades. Se ha querido mantener la uniformidad en las tareas que son muy parecidas. Los pasos son los siguientes:

1. Antes de enviar la petición se comprueba que el campo nombre no esta vacío, que existe al menos una actividad para este reto y que se ha seleccionado un diploma.
2. Se recibe una petición de `POST` a `/challenges/register` que contiene el nombre del reto, la descripción, las actividades que lo componen y el diploma asociado como parámetros en el cuerpo de la petición.
3. Se comprueba que no existe ningún reto con el mismo nombre que la presente en `req.body.challengeName`. Si no es así, se vuelve a generar la vista mostrando como error que ya existe un reto con el mismo nombre.
4. Se crea un objeto `challenge` con los parámetros de `req.body` y se llama al método `Challenge.createChallenge`, que guarda en la base de datos el nuevo reto.
5. Se redirige al usuario hacia `/challenges`, donde aparecerá el nuevo reto creado.

Le controlador que se encarga de cursar las peticiones y generar una vista adecuada de los retos se encuentra en la ruta `/routes/challenges.js`. En la figura 4.16 se muestra una pequeña porción del fichero.

```
/* GET home page. */
router.get('/', auth.ensureAthentication, function(req, res) {
  Challenge.getChallenges(function(err, challenges) {
    if (err) throw err;
    else {
      res.render('challenges/challenges', {
        challenges: challenges});
    }
  });
});

// GET register page.
router.get('/register', auth.ensureAthentication, function(req, res) {
  Diploma.getDiplomas(function(err, diplomas) {
    if (err) throw err;
    Activity.getActivities(function(err, activities) {
      if (err) throw err;
      res.render('challenges/register', {
        diplomas: diplomas,
        activities: activities
      });
    });
  });
});
```

Figura 4.16: Modelo de `Challenge`

4.3.2. Funcionalidades adicionales

4.3.2.1. Ahorcado

Una vez desarrolladas las funcionalidades básicas, llego el momento de el primer reto de la plataforma.

Para adaptar el juego del ahorcado se busco varias implementaciones de este funcionando que usaran la combinación JavaScipt para la lógica del juego, HTML para la visualización del mismo y CSS para el estilo. Se decidió que era necesario que dicha versión permitiese la modificación y redistribución del juego sin ninguna restricción.

Se dio con una versión que cumplía con todos estos requisitos, pues esta distribuida bajo la licencia MIT. Dicha implementación puede encontrarse en la dirección `https://codepen.io/cathydutton/pen/ldazc`.

Antes de proceder con la adaptación,fue necesario especificar una serie de requisitos que debían cumplirse en la versión final:

- Debía mostrar una imagen en movimiento junto al lienzo donde se dibujará el ahorcado a medida que no se acierten letras de la frase oculta.
- Debía permitir guardar información sobre si la actividad había sido completada por parte del usuario.
- Debía permitir trabajar con frases, puesto que la versión que se encontró solo contemplaba el uso de palabras como mensajes ocultos.
- Debía mostrar una barra de progreso en la que se representara el porcentaje de actividades del reto completadas.
- No era necesario mostrar ninguna pista acerca de las frases ocultas, puesto que son que resultan familiares para el usuario.

Una vez establecidas las restricciones, se procedió a adaptar el juego empezando por la interfaz gráfica. Para ello, se tuvieron en cuenta los diseños descritos en la sección 4.2.2.

La primera pagina antes del comenzar el reto debía mostrar cinco imágenes, las cuales el usuario debía relacionar con cinco frases. Estos aspectos de diseño se describieron en la sección 4.2.2.1. Se considero que no había ninguna asociación preestablecida, puesto que el dolor es algo subjetivo que puede variar de un usuario a otro. Así pues, el usuario puede comenzar el reto cuando crea oportuno, puesto que hay ninguna restricción.

La figura 4.17 muestra el resultado final del diseño de la pagina previa a la primera actividad.

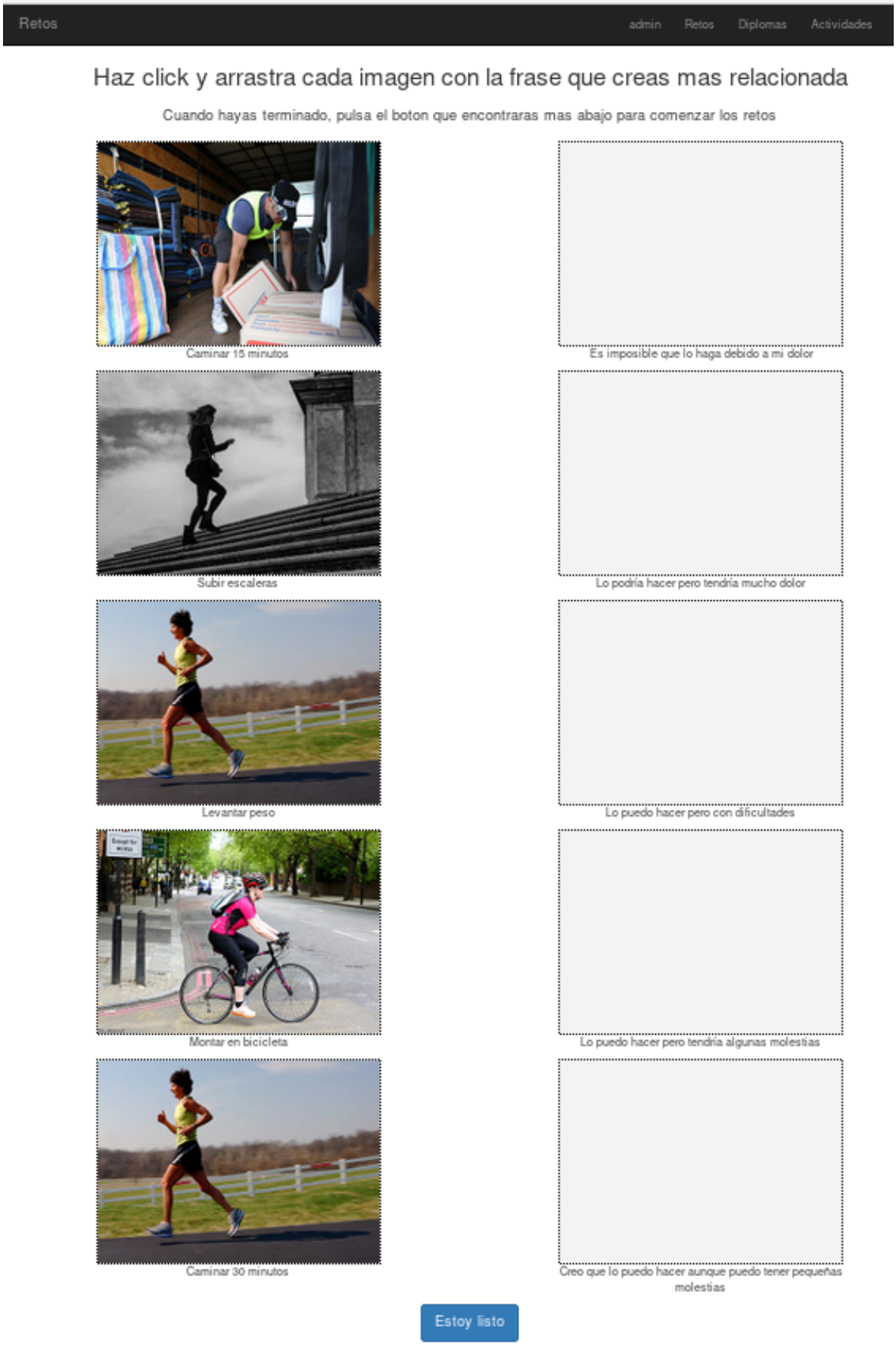

Figura 4.17: Diseño final pagina inicial del reto del ahorcado

Tras tener la vista previa al inicio del reto lista, fue el momento de empezar con la parte central del reto, el juego en si. Para ello, se adaptó al diseño acordado en la fase previa de desarrollo la nueva versión creada a partir del *fork* del ahorcado de Cathy Dutton. La figura 4.18 muestra el diseño final.

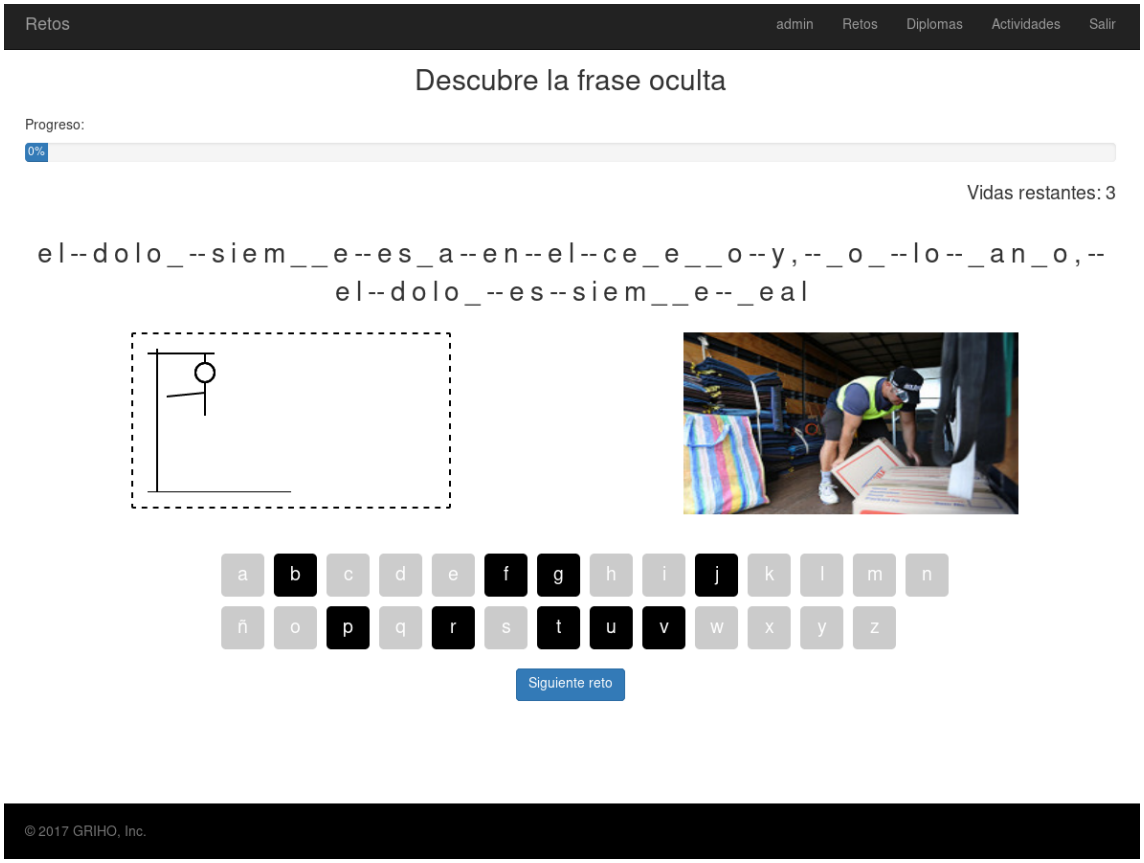

Figura 4.18: Diseño final pagina del juego del reto del ahorcado

Tras tener la interfaz lista, se prosiguió a adaptar el código. El código base que se usa para cada una de las pantallas se encuentra en la ruta `/public/js/hangman.js`. Hubo que resolver un problema que no se había planteado hasta que necesito guardarse información en la base de datos cuando el usuario conseguía descifrar la frase.

Como era necesario mandar una petición POST al servidor cuando esto ocurría pero no quería usarse ninguna `form` que hiciese que el servidor nos redirigiese a otra ruta, se uso AJAX.

De esta forma, una vez se completaba el reto, se mandaba una petición al servidor sin alterar la pagina en la que el usuario se encontraba. En la figura 4.19 puede verse la función que se encarga de esto y que se encuentra en el código que ejecuta el navegador del cliente.

```
<script>
$('#win').on('shown.bs.modal', function () {
  var data = {
    userId: '{{userId}}',
    activityId: '{{activityId}}'
  }

  $.ajax({
    url: '/challenges/hangman/fleet',
    type: 'POST',
    dataType: 'json',
    data: data
  });
});

$('#win').on('hidden.bs.modal', function () {
  var user = {
    userId: '{{userId}}'
  }

  $.ajax({
    url: '/challenges/hangman',
    type: 'POST',
    dataType: 'json',
    data: user
  });
});
</script>
```

Figura 4.19: Función AJAX

La primera función es llamada cuando se muestra un menú contextual en el momento en que se consigue descifrar toda la frase. Cuando el servidor recibe esta petición, se encarga de guardar en la base de datos la actividad que el usuario ha completado con éxito. Esta información se guarda en un modelo que asocia cada usuario con las actividades que ha superado. Este modelo puede verse en la figura 4.20.

```
// UserActivities Schema
var UserActivitiesSchema = mongoose.Schema({
  user: {
    type: Schema.ObjectId,
    ref: 'User',
    required: true
  },
  activities: [{
    type: Schema.ObjectId,
    ref: 'Activity'
  }]
});

var UserActivities = module.exports = mongoose.model('UserActivities', UserActivitiesSchema);
```

Figura 4.20: Modelo de `UserActivities`

La segundo función de la figura 4.19 es llamada cuando el menú contextual desaparece del primer plan. Se encarga de comprobar si al completar dicha frase, el usuario ha completado todas las actividades del reto. Si es así, se guarda en los objetos `UserChallenges` y `UserDiplomas` de la base de datos con el `id` del usuario actual los desafíos y diplomas en el que se estaba participando.

```
// UserDiplomas Schema
var UserChallengesSchema = mongoose.Schema({
  user: {
    type: Schema.ObjectId,
    ref: 'User',
    required: true
  },
  challenges: [{
    type: Schema.ObjectId,
    ref: 'Challenge'
  }]
});

var UserChallenges = module.exports = mongoose.model('UserChallenges', UserChallengesSchema);
```

Figura 4.21: Modelo de `UserChallenges`

```
// UserDiplomas Schema
var UserDiplomasSchema = mongoose.Schema({
  user: {
    type: Schema.ObjectId,
    ref: 'User',
    required: true
  },
  diplomas: [{
    type: Schema.ObjectId,
    ref: 'Diploma'
  }]
});

var UserDiplomas = module.exports = mongoose.model('UserDiplomas', UserDiplomasSchema);
```

Figura 4.22: Modelo de `UserDiplomas`

Dichos modelos pueden verse en las figuras 4.21 y 4.25. Ambos modelos son creados cuando el usuario se registra en el sistema.

Gracias a esa información almacenada sobre cada usuario, es posible recuperar en el perfil la información que la las figuras 4.12, 4.12 y 4.12 muestran.

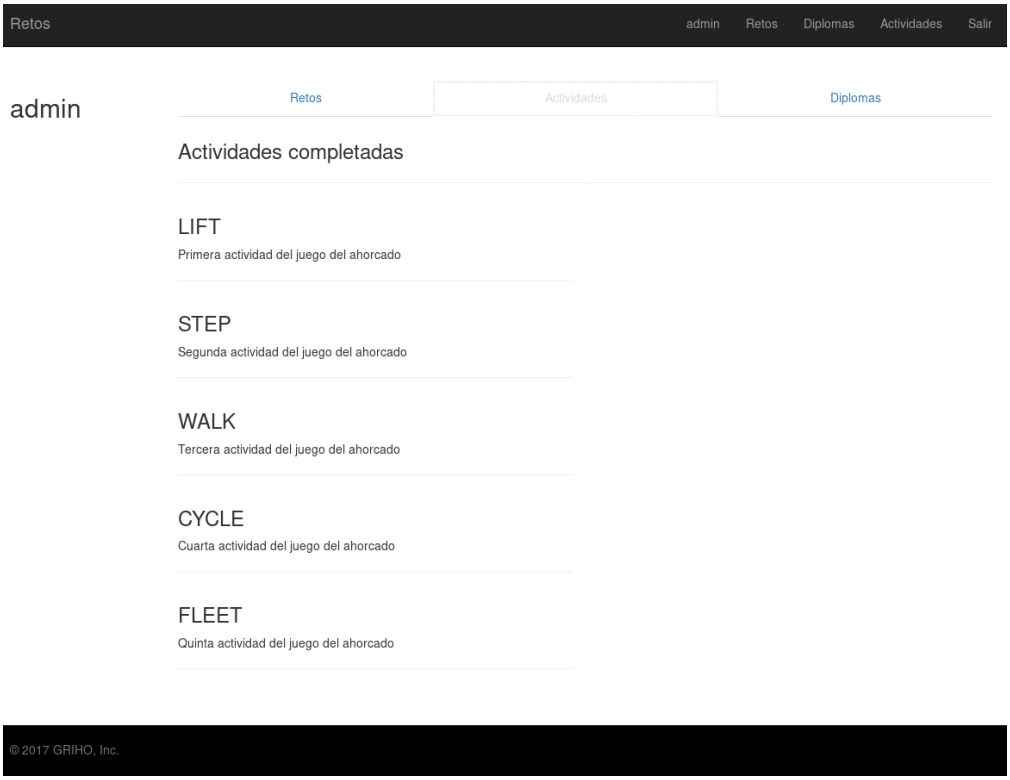

Figura 4.23: Actividades completadas

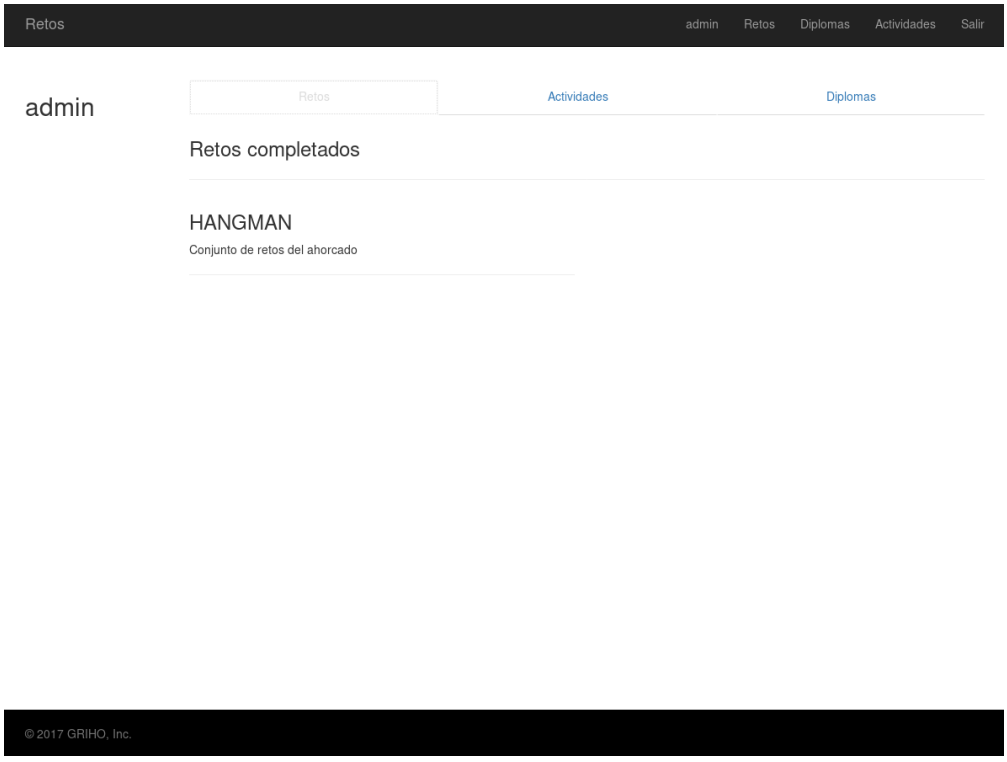

Figura 4.24: Retos completadas

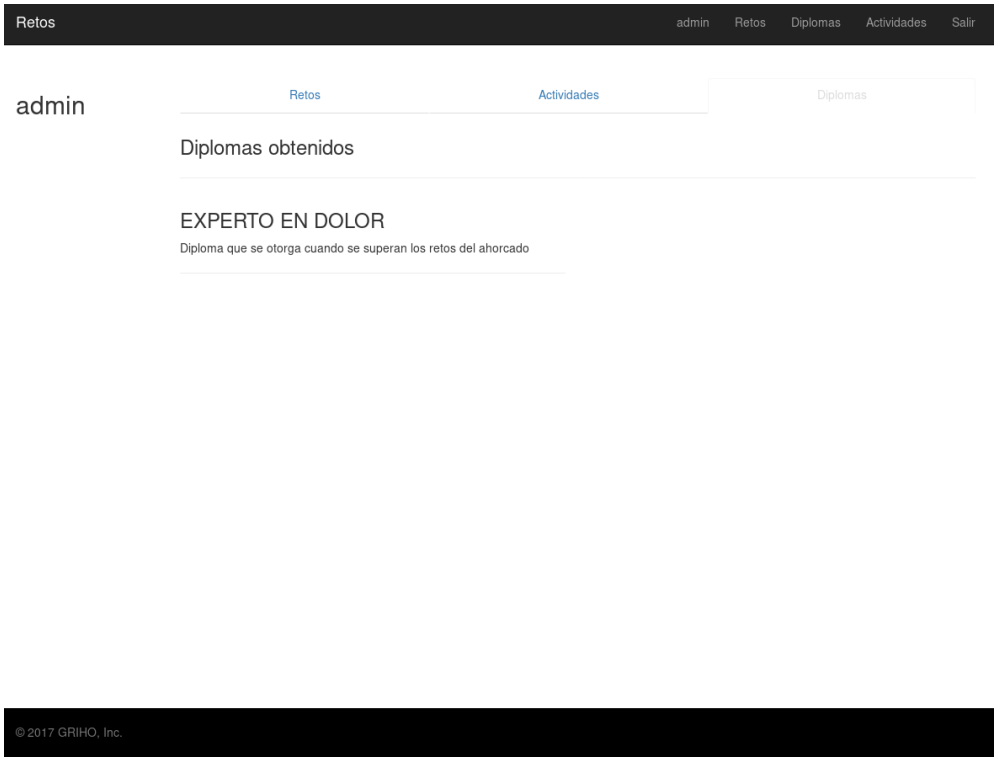

Figura 4.25: Diplomas conseguidos

4.3.2.1.1 Animaciones

Para la creación de las animaciones mostradas en cada uno de los retos, fue necesario utilizar una herramienta de edición de gráficos vectoriales como Inkscape, explicada en el apartado 3.1.4.2.

Se prosiguió con la creación de un modelo para cada una de las actividades. Más tarde, con cada uno de estos modelos se crearon las respectivas *sprite sheets*. La figura 4.26 muestra un frame de una de estas *sprite sheets*.

Figura 4.26: *Frame* de *sprite sheet*

Una vez generadas el conjunto de imágenes, se procedió a animar dichas *sprite sheets* por medio de CSS. La figura 4.27 muestra el código de la animación que simula la actividad de caminar.

```
.wakling {
    height: 200px;
    width: 209.75px;
    background-image: 'images/walk.png';
    animation: walk 1.5s steps(34) infinite;
}

@keyframes walk {
    100% {background-position: -7130px;}
}
```

Figura 4.27: Funciones de animación de movimiento

4.4. Implantación

En este apartado se detallarán los requisitos y pasos necesarios para tener la aplicación lista para su uso.

4.4.1. Requisitos

Antes de poder ejecutar la aplicación es necesario tener instalados en la maquina que va a hacer de servidor el siguiente software:

- Node.js.
- MongoDB.

4.4.2. Procedimiento

Tras instalar el software mencionado en el apartado anterior, es necesario descargar la aplicación de `https://github.com/gerardmr90/challenges`. Una vez descargado y descomprimido el archivo, hay que ejecutar los siguientes comandos:

- `npm install`
- `npm start`

En este punto, la aplicación estará funcionando correctamente, pero se necesitara crear un nuevo usuario para acceder a ella. Una vez iniciada sesión habrá que hacer seguir estos últimos pasos:

1. Crear las actividades para el ahorcado con los nombres y ordenes siguientes:
   a) Primera actividad con nombre *"lift"*.
   b) Segunda actividad con nombre *"step"*.

   c) Tercera actividad con nombre *"walk"*.
   d) Cuarta actividad con nombre *çycle"*.
   e) Quinta actividad con nombre *"fleet"*.

2. Crear un nuevo diploma.
3. Crear el reto con nombre *"hangman"* con las actividades y el diploma creados en los pasos anteriores.

Tras esto, la aplicación ya sera completamente funcional y podrá empezar a usarse en un entorno real.

Capítulo 5: Finalización

En este capitulo final, se explicaran las conclusiones que se han obtenido tras llevar a cabo el proyecto. También se indicaran ciertas ampliaciones o mejoras futuras.

5.1. Conclusiones

Tras la concluir el proyecto, se pueden sacar varias conclusiones. La primera de ellas es que el tiempo es un factor vital en el diseño y desarrollo de cualquier proyecto software. De él depende el éxito o fracaso del mismo.

La segunda conclusión que se ha obtenido es que la fase de análisis y diseño son las fases mas importantes y a las que, personalmente, no les había dado la importancia merecida hasta la realización de este trabajo. Esto es debido a que un buen análisis y un buen diseño hacen que en la fase de desarrollo se minimicen los imprevistos que se puedan sufrir.

El hecho de tener que realizar el proyecto en un campo que no tenia demasiado experiencia, como es el desarrollo web, me ha hecho tener que aprender tecnologías totalmente desconocidas para mi hasta este momento. Personalmente, he disfrutado creando la aplicación desde cero. Sin embargo, me he dado cuenta que cuanto más aprendo, mas me doy cuenta que no se nada, un pensamiento recurrente en el ultimo año.

Por ultimo, conocer un poco a cerca de la gente que sufre dolor crónico ha hecho empatizar con la gente que lo sufre. Si con la aplicación se consigue ayudar, estaré más que satisfecho.

5.2. Trabajos futuros

Tras la realización del proyecto, seria necesario analizar las posibles ampliaciones de funcionalidad:

- Añadir tablas de clasificación en que se mostrase el usuario más experto en su dolor, de forma que se impulsase la competición y se despertase en los usuarios el sentimiento de querer progresar.

- Permitir que los usuarios pudiesen tomarse una foto antes de comenzar los retos, de forma que las animaciones que se muestra, tuviesen la cara de cada usuario, logrando así un vinculo mas fuerte entre juego y usuario.
- Permitir compartir cada uno de los retos y diplomas conseguidos en las redes sociales

Bibliografía

[1] Atom. Atom Editor. https://atom.io/, 2017. [Online; accedido el 30/8/2017].

[2] Bootstrap. Twitter Bootstrap. http://getbootstrap.com/, 2017. [Online; accedido el 29/8/2017].

[3] Pencil Evolus. Pencil. https://pencil.evolus.vn/, 2017. [Online; accedido el 30/8/2017].

[4] Express. Express. http://expressjs.com/, 2017. [Online; accedido el 28/8/2017].

[5] Virginia Gaitán. Gamificación: El aprendizaje divertido. http://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/, 2017. [Online; accedido el 6/9/2017].

[6] Git. Git. https://git-scm.com/, 2017. [Online; accedido el 30/8/2017].

[7] GitHub. GitHub. https://github.com, 2017. [Online; accedido el 30/8/2017].

[8] Handlebars. Handlebars. http://handlebarsjs.com/, 2017. [Online; accedido el 29/8/2017].

[9] Inkscape. Inkscape. https://inkscape.org/es/, 2017. [Online; accedido el 5/9/2017].

[10] JavaScript. JavaScript. https://www.javascript.com/, 2017. [Online; accedido el 29/8/2017].

[11] MongoDB. MongoDB. https://www.mongodb.com/, 2017. [Online; accedido el 28/8/2017].

[12] Mongoose. Mongoose. http://mongoosejs.com/index.html, 2017. [Online; accedido el 29/8/2017].

[13] Mustache. Mustache. https://mustache.github.io/, 2017. [Online; accedido el 29/8/2017].

[14] Regina Nelson. Gamification Mechanics, Dynamics and Components. https://www.uwplatt.edu/ttc/gamification-mechanics-dynamics-and-components, 2017. [Online; accedido el 6/9/2017].

[15] Node.js. Node.js. `https://nodejs.org/es/`, 2017. [Online; accedido el 28/8/2017].

[16] OAuth. OAuth. `https://oauth.net/`, 2017. [Online; accedido el 28/8/2017].

[17] Passport. Passport. `http://passportjs.org/`, 2017. [Online; accedido el 28/8/2017].

[18] John Resig. jQuery. `https://jquery.com/`, 2017. [Online; accedido el 30/8/2017].

[19] W3C. CSS. `https://www.w3.org/Style/CSS/Overview.en.html`, 2017. [Online; accedido el 29/8/2017].

[20] Kevin Werbach and Dan Hunter. *For the Win*. Wharton Digital Press, Philadelphia, 2012.

[21] Wikipedia. Ahorcado (Juego). `https://es.wikipedia.org/wiki/Ahorcado_(juego)`, 2017. [Online; accedido el 28/8/2017].

[22] Wikipedia. AJAX. `https://es.wikipedia.org/wiki/AJAX`, 2017. [Online; accedido el 30/8/2017].

[23] Wikipedia. Framework. `https://es.wikipedia.org/wiki/Framework`, 2017. [Online; accedido el 28/8/2017].

[24] Wikipedia. GNU Affero General Public License. `https://es.wikipedia.org/wiki/GNU_Affero_General_Public_License`, 2017. [Online; accedido el 28/8/2017].

[25] Wikipedia. GNU General Public License. `https://es.wikipedia.org/wiki/GNU_General_Public_License`, 2017. [Online; accedido el 30/8/2017].

[26] Wikipedia. HTML. `https://es.wikipedia.org/wiki/HTML`, 2017. [Online; accedido el 29/8/2017].

[27] Wikipedia. Licencia MIT. `https://es.wikipedia.org/wiki/Licencia_MIT`, 2017. [Online; accedido el 28/8/2017].

[28] Wikipedia. Middleware. `https://es.wikipedia.org/wiki/Middleware`, 2017. [Online; accedido el 28/8/2017].

[29] Wikipedia. Modelo-vista-controlador. `https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador`, 2017. [Online; accedido el 1/9/2017].

[30] Wikipedia. NoSQL. `https://es.wikipedia.org/wiki/NoSQL`, 2017. [Online; accedido el 28/8/2017].

[31] Wikipedia. Plugins (Informática). `https://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)`, 2017. [Online; accedido el 28/8/2017].

[32] Wikipedia. Protocolo HTTP. https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto, 2017. [Online; accedido el 28/8/2017].

[33] Wikipedia. Script. https://es.wikipedia.org/wiki/Script, 2017. [Online; accedido el 28/8/2017].

[34] Wikipedia. Sprite (videojuegos). https://es.wikipedia.org/wiki/Sprite_(videojuegos), 2017. [Online; accedido el 5/9/2017].